

a1
22. (New) The method of claim 21, wherein said executing entity executes said downloaded second portion of the code directly from said second executable memory component.

A marked-up version of the changes made to the specification and claims by the current amendment is attached, with the header "VERSION WITH MARKINGS TO SHOW CHANGES MADE".

REMARKS

Reconsideration of the above-identified patent application in view of the amendments above and the remarks following is respectfully requested.

Claims 1-9 are in this case. Claims 1-3 and 6-8 have been rejected under § 102(b). Claims 1-9 have been rejected under § 102(e). Claims 1, 4, 6, 7 and 9 have been objected to. Independent claims 1, 4 and 6 and dependent claims 7 and 9 have been amended. New claims 10-22 have been added.

The claims before the Examiner are directed toward a system and method for directly executing code stored in a non-executable memory component. One or more executable memory components are provided to buffer a portion of the code to an executing entity. When more than one executable memory component is used, then while one executable memory component functions as a buffer of one portion of the code, another portion of the code is downloaded from the non-executable memory component to another executable memory component in order to guarantee continuous access to the code by the executing entity.

§ 102(b) Rejections – Honma et al. ‘529 and Rhee ‘673

§ 102(e) Rejections – Devanagundy et al. ‘384 and Pashley et al. ‘506

The Examiner has rejected claims 1-3, 6 and 8 under § 102(b) as being anticipated by Honma et al., US Patent No. 5,606,529 (henceforth, “Honma et al. ‘529”). The Examiner also has rejected claims 1-3, 6 and 7 under § 102(e) as anticipated by Devanagundy et al., US Patent No. 6,148,384 (henceforth, “Devanagundy ‘384”). The Examiner also has rejected claims 1-6, 8 and 9 under § 102(e) as being anticipated by Pashley et al., US Patent No. 6,418,506 (henceforth, “Pashley et al. ‘506”). The Examiner also has rejected claims 1-3, 6 and 7 under § 102(b) as being anticipated by Rhee, US Patent No. 5,608,673 (henceforth, “Rhee ‘673”). The Examiner’s rejection is respectfully traversed.

All four prior art patents cited by the Examiner teach systems in which nonvolatile memory is buffered via volatile memory. Honma et al. ‘529 teach a storage device in which data to be written to flash memories 2 or 5 or read from flash memories 2 or 5 by a host device is cached in a cache memory 3. Devanagundy et al. ‘384 teach an integrated circuit, such as a host adapter 140 for connection to a system bus 120 of a host computer 110, in which data are read by host computer 110 from a SEEPROM 144 via registers SEEDAT0 and SEEDAT1 of control registers 350. Pashley et al. ‘506 teach a memory device 100 in which data to be written to a flash array 103 or read from flash array 103 by a processor 104 are cached in a RAM write buffer array 101. Rhee ‘673 teaches a NAND-type flash memory integrated circuit card in which data to be written to NAND flash memory components 1-8 or read from NAND flash memory components 1-8 by an electronic product are cached in a buffer 51, and in which data to be written to NAND flash memory components 9-16 or read

from NAND flash memory components 9-16 by an electronic product are cached in buffers 52 and 53.

As best understood, the Examiner's rejection of independent claims 1, 4 and 6 is based on the premise that the data cached by Honma et al. '529, Devanagundy et al. '384, Pashley et al. '506 and Rhee '673 could include executable code. In fact, the only prior art reference cited by the Examiner that could conceivably be construed as anticipating the present invention is Honma et al. '529, who refer in passing, in column 2 lines 30-33, to "the temporary saving of a program code existing on a main storage to the non-volatile storage medium and the reading of the program code from the non-volatile storage medium". Pashley et al. '506 refers only to buffering "data" that are stored in flash array 103. Similarly, Rhee '673 refers only to "data" (illustrated as "Din 0" through "Din 255" in Figure 2F) that are written to NAND flash memory components 1-16 or "data" (illustrated as "Dout N" through "Dout N+L" in Figure 4F) that are read from NAND flash memory components 1-16. Devanagundy et al. '384 state explicitly that the data that they store in SEEPROM 144 are "critical configuration information and non-critical information" (column 4 lines 32-33), and not executable code. Even so, as best understood, what Honma et al. '529 have in mind is not the present invention, as recited in independent claims 1, 4 and 6, but rather the prior art method of downloading executable code from a nonvolatile memory to the RAM of the host device that is large enough to accommodate the full address range of the executable code. This is clear from the use by Honma et al. '529 of "Fixed Block Architecture", the standard format of hard disk storage devices (column 2 lines 14-18), for "data access from the host device" (column 4 line 67 to column 5 line 1). As is well known, hard disk storage devices are serial access devices, and so are non-executable memory devices. Executable

code stored on a hard disk storage device would have to be read to the RAM of a host device for execution.

Furthermore, the present invention is not even obvious from the prior art references cited by the Examiner. As noted above, the only prior art reference with even a hint or suggestion of buffering executable code to a host device is Honma et al. '529. But one ordinarily skilled in the art would be led by a study of Honma et al. '529 only to the prior art method, discussed above, of downloading executable code from a nonvolatile memory to a RAM of the host device that is large enough to accommodate the full address range of the executable code, and not to the present invention. The present invention is based on the realization that the combination of a non-executable memory component, such as NAND flash memory 14, that is large enough to accommodate the full address range of executable code, and an executable memory component, such as SRAM 10, that is too small to accommodate the full address range of the executable code, could be used for direct execution of code by an executing entity 14, provided that provision is made for compensating for download latency when code not currently stored in the executable memory component is requested by executing entity 14. The issue of download latency is not addressed at all by Honma et al. '529. There is neither a hint nor a suggestion anywhere in Honma et al. '529 of any need to compensate for download latency. Thus, the combination of a non-executable memory component for storing system code and an executable memory component for operating as a memory buffer for executing the code (as opposed to operating as a buffer for merely transferring the code to the RAM of the host device), as recited in independent claim 1, would not be obvious to one ordinarily skilled in the art from a study of Honma et al. '529. Similarly, the combination of a non-executable memory component for storing code and a plurality of executable

memory components that prevent memory lockage (*i.e.*, accounting for download latency) during downloads of the code, as recited in independent claim 4, would not be obvious to one ordinarily skilled in the art from a study of Honma et al. '529; and the combined steps of storing executable code in a non-executable memory and buffering an execution of the executable code in an executable memory (as opposed to buffering the transfer of the code to the RAM of a host device), as recited in independent claim 6, would not be obvious to one ordinarily skilled in the art from a study of Honma et al. '529.

With independent claims 1, 4 and 6 allowable in their present form, it follows that claims 2, 3, 5 and 7-9, that depend therefrom, also are allowable.

Although claims 7 and 9 are allowable by virtue of depending from claim 1, Applicant now takes the liberty of presenting more reasons why these claims are allowable over the prior art cited against them by the Examiner.

Claim 7 recites the step of supplying a busy signal when requested contents of the non-executable memory are not available, such that the executing entity delays the read cycle until the contents are available. The Examiner has cited Devanagundy et al. '384 as anticipating claim 7, on the grounds that Devanagundy et al. '384 teach setting a busy bit while registers SEEDAT0 and SEEDAT1 are being loaded, and polling the busy bit by host computer 110 to find out when the loading of registers SEEDAT0 and SEEDAT1 is complete. The busy bit of Devanagundy et al. '384, which must be actively polled by host computer 110, is quite unlike the busy signal of the present invention, which is received passively by the executing entity. Therefore, the busy signal of the present invention is not anticipated by the busy bit of Devanagundy et al. '384.

Claim 9 recites the steps of loading the executable code into one executable memory buffer while maintaining another executable memory buffer that is accessible to and executable by the executing entity. The Examiner has cited Pashley et al. '506 as anticipating claim 9, on the grounds that Pashley et al. '506 teach, in column 8 lines 59-64, that data can be simultaneously read from flash array 103 and written to RAM array 101 by an external device. This is not what is recited in claim 9. Claim 9 recites executing the code stored in one executable memory buffer by the executing entity (an external device) while writing other code, not from the executing entity to another executable memory device, but from the non-executable memory to the other executable memory buffer.

Objections to the Claims

The Examiner has objected to informalities in claims 1, 4, 6, 7 and 9. These claims now have been amended to rectify the informalities.

In claims 1, 4 and 6, the periods have been removed from the Roman numerals and the Roman numerals have been enclosed in parentheses.

In claims 1 and 4, the internal capitalization and the internal periods have been removed.

In claim 7, "executable entity" has been replaced with --executing entity--.

In claim 9, the Roman numerals have been replaced with lower case Latin letters in parentheses.

Other Amendments to the Claims

In claim 4 the description of the executable memory components has been brought into closer conformity with the specification. Specifically, the executable memory components now are described as alternating as memory buffers, thereby

preventing memory lockage. Support for this amendment is found in the specification on page 10 line 22 through page 11 line 1:

...the download operation will load the requested content to one SRAM buffer...while the other SRAM buffer **20** remains accessible and executable.

In addition, the redundant word “multiple” has been deleted.

In claim 7, the lower case Latin letters that identify the steps have been replaced with Roman numerals, for consistency with claim 6.

In claim 9, the italics have been replaced with regular font. In addition, an obvious inadvertent error has been corrected: “said host system”, which lacks antecedent basis, has been corrected to “said executing entity”. Support for this amendment is found in the specification on page 8 line 2, where the host system is identified with executing entity **12**.

Amendments to the Specification

The paragraph beginning on page 8 line 10 has been amended to ensure that all reference numerals are in boldface. In the paragraph beginning on page 10 line 8, an inadvertent typographical error (reference numeral “**30**” instead of reference numeral **–12--**) has been corrected. No new matter has been added.

New Claims

New claims 10-22 have been added.

New independent claim 10 recites the device of the present invention as a combination of a non-executable memory component for storing code and at least one executable memory component for presenting at least a portion of the stored code to an executing entity in a manner that enables the executing entity to execute that portion of the stored code directly from the executable memory component, without

reciting the executing entity/host system as a component of the present invention. Support for the non-executable memory is found in the specification in non-executable memory component 14 of Figure 1. Executing code “directly” from a memory component is defined in the specification on page 1 line 14 through page 2 line 2:

The ability to execute code directly from a memory device usually requires the following characteristics from the memory device:

1. Standard memory interface, which includes address bus, data bus and a few more control signals...The executing entity...determines the address of the required code..., executes a given cycle...and expects that the required code will appear on the data bus after a short period of time...
2. Full visibility of the whole memory space of the memory device. This means that the executing entity can change the address values to any valid state (within the memory space of the device) and still get the required code after the same period of delay. This behavior is also known as Random Access ability. (emphasis added)

That the present invention includes an executable memory component that enables “direct” execution of code stored therein is stated on page 8 lines 10-12 of the specification with regard to executable memory component 10 of Figure 1:

The 1KB of SRAM is executable and satisfies the requirements of an executable memory (Random Access and standard memory interface).

New dependent claim 11 states explicitly that the non-executable memory component and the executable memory component(s) are separate from the host system. Support for this limitation is found in the specification in Figure 1, which shows executable memory component (SRAM) 10 and non-executable memory component (NAND flash) 14 as separate from executing entity 12. Recall that page 8 line 2 of the specification identifies the host system with executing entity 12.

New dependent claim 12 limits the present invention to also include a mechanism for guaranteeing availability, in one of the executable memory

component(s), of code requested by the executing entity. Support for this limitation is found in the specification on page 9 lines 1-4:

In order to enable proper control over the SRAM buffer **10** contents, the implementation includes download online algorithms, for enabling fast downloading of data from the NAND flash to the SRAM buffer as well as for guaranteeing the availability of the requested information in the executable buffer/s.

and on page 9 lines 17-18:

The device of the present invention manages at least one algorithm to guarantee availability of the requested information in the executable buffer/buffers.

One example of the “mechanism” recited in new dependent claim 12 is illustrated in Figure 2 as download logic **22**.

New dependent claim 13 requires the present invention to have a plurality of the executable memory components, such that while one executable memory component is presenting a first portion of the stored code to the executing entity, a second portion of the stored code is being downloaded to another executable memory component. Support for this limitation is found in the specification on page 10 line 19 through page 11 line 1:

A further preferred embodiment of the present invention describes a dual or multiple SRAM buffer **20**. This buffer, or set of buffers, as can be seen in figure 2 can be used in order to prevent the memory from being locked for accesses during download operations. In this case the download operation will load the requested content to one SRAM buffer...while the other SRAM buffer **20** remains accessible and executable.

New dependent claim 14 limits the executable memory component(s) to being too small to accommodate all of the stored code at once. This limitation is implicit in the description on page 9 lines 9-16 of the specification:

As long as the requested address (requested by the executing entity **12**) is included in the current SRAM buffer contents, the device will satisfy the required code immediately, from the buffer, and will not impose any content changes.

When the requested address is not contained within the current SRAM buffer contents, the download algorithms initiate a download operation from the required location of the NAND flash **14** to the SRAM buffer **10**. Upon completion of the download operation, the required information is available in an executable manner inside the SRAM buffer **10**.

After the initial download of code to SRAM buffer **10**, the only reason that the requested address would not be contained within the current SRAM buffer contents is that SRAM buffer **10** is too small to accommodate all of the code at once.

New independent claim 15 recites the method of the present invention as the steps of storing code in a non-executable memory component, downloading at least a portion of the code to an executable memory component, and executing the downloaded code, by an executing entity of a host system, without reference to “emulating executable functions”. In addition, the executable memory component must be separate from the host system. Support for the step of storing code in a non-executable memory is found in the specification in non-executable component **14** of Figure 1. Support for the step of downloading code from the non-executable memory component to the executable memory component is found in the specification on page 8 lines 15-17:

Data from the NAND flash array **14** is downloaded into the SRAM **10**, such that the data inside the SRAM **10** is a copy of a portion of the NAND flash **14** content.

and on page 9 line 13-14:

...the download algorithms initiate a download operation from the required location of the NAND flash **14** to the SRAM buffer **10**.

Support for the step of executing the downloaded code by an executing entity of a host system is found in the specification on page 7 line 18 through page 8 line 2:

...there is provided an executing entity **12**...for executing code and processing data...The final primary component of the present invention is a small amount of fully mapped executable memory **10**...This executable memory acts as a buffer for the purpose of code

execution, and is configured to have full visibility of a host system (the executing entity 12).

and on page 8 lines 11-12:

A CPU or executing entity 12 can therefore execute any location within the range of SRAM 10.

Support in the specification for the executable memory component being separate from the host system is discussed above in the context of new claim 11.

New dependent claim 16 requires the executing entity to execute the downloaded code directly from the executable memory component. Support in the specification for this limitation is discussed above in the context of new claim 10.

New dependent claim 17 requires that only a portion of the code stored in the non-executable memory component be downloaded to the executable memory component. Support for this limitation is found in the specification on page 8 lines 13-18:

...the 1KB of SRAM 10 contains up to 1KB from the NAND flash array 14 (out of the 8MB of the total NAND capacity). Data from the NAND flash array 14 is downloaded into the SRAM 10, such that the data inside the SRAM 10 is a copy of a portion of the NAND flash 14 content. (emphasis added)

New dependent claim 18 adds to new independent claim 15 the steps of requesting code to be executed, by the executing entity, after the first downloading, and then, if the requested code is not in the executable memory component, downloading a second code portion that includes the requested code, while suspending activity of the executing entity. Support for the executing entity requesting code to be executed is found in the specification on page 9 line 9, which states that the “requested data/address” of page 9 lines 7-8 was requested by executing entity 12. That the context of the request is one in which code has already been

downloaded to the executing memory component is clear from page 9 lines 9-10 of the specification:

As long as the requested address...is included in the current SRAM buffer contents...

Support for downloading a second code portion if the requested code is not in the executable memory component is found in the specification on page 9 lines 12-14:

When the requested address is not contained within the current SRAM buffer contents, the download algorithms initiate a download operation from the required location of the NAND flash **14** to the SRAM buffer **10**.

Support for suspending activity of the executing entity during the second download is found in the specification on page 9 line 20 through page 10 line 2:

...in cases when the required data...is not within the SRAM current range, the data must be downloaded from the NAND to the SRAM. This operation takes time...during which the required addresses cannot return the required code...The provided busy signal therefore alerts the host system to cause the host system to cease data requests until downloading of the data is complete. (emphasis added)

and on page 10 lines 12-17 as now amended:

...while the memory device is in the process of downloading the required code for execution, it is required to supply a “busy signal” **26** to the executing entity **12** in order to notify that the required code is not yet available. The executing entity **12** should use the “busy signal” **26** in order to hold off the execution attempt until the memory device is ready and able to supply the required code. (emphasis added)

New dependent claim 19 requires that the act of suspending the activity of the executing entity during the second download include supplying a busy signal to the executing entity. Support for this limitation is found in the specification on page 9 lines 18 through page 10 line 2:

The device supplies a busy signal in cases when the information is not yet available...The provided busy signal therefore alerts the host system to cause the host system to cease data requests until downloading of the data is complete.

and on page 10 lines 12-15:

...while the memory device is in the process of downloading the required code for execution, it is required to supply a “busy signal” 26 to the executing entity 12 in order to notify that the required code is not yet available.

New dependent claim 20 adds to new dependent claim 16 the steps of downloading a second code portion to a second executable memory component and having the executing entity execute the second downloaded code portion. Support for downloading a second code portion to a second executable memory component is found in the specification on page 10 line 22 through page 11 line 1:

...the download operation will load the requested content to one SRAM buffer...while the other SRAM buffer 20 remains accessible and executable.

Support for having the executing entity execute the second downloaded code portion is found in the specification on page 11 lines 2-3:

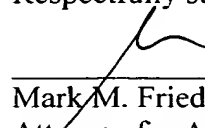
...with proper support of the download state machine, it is possible to support code execution from one or more buffers...(emphasis added)

New dependent claim 21 requires the second executable memory component to be separate from the host system. Support in the specification for this limitation is discussed above in the context of new claim 11.

New dependent claim 22 requires the executing entity to execute the second downloaded code portion directly from the second executable memory component. Support in the specification for this limitation is discussed above in the context of new claim 10.

In view of the above amendments and remarks it is respectfully submitted that independent claims 1, 4, 6, 10 and 15, and hence dependent claims 2, 3, 5, 7-9, 11-14 and 16-22 are in condition for allowance. Prompt notice of allowance is respectfully and earnestly solicited.

Respectfully submitted,



Mark M. Friedman
Attorney for Applicant
Registration No. 33,883

Date: December 8, 2002

VERSION WITH MARKINGS TO SHOW CHANGES MADE

In the specification:

The paragraph beginning on page 8 line 10 has been amended as follows:

1. The 1KB of SRAM is executable and satisfies the requirements of an executable memory (Random Access and standard memory interface). A CPU or executing entity **12** can therefore execute any location within the range of the SRAM **10**. According to the present invention, at any given time, the 1KB of SRAM **10** contains up to 1KB from the NAND flash array **14** (out of the 8MB of the total NAND capacity). Data from the NAND flash array **14** is downloaded into the SRAM ~~10~~**10**, such that the data inside the SRAM ~~10~~**10** is a copy of a portion of the NAND flash **14** content. In this way, the SRAM **10** functionality is emulated for the NAND **12** content, which enables the NAND **12** content within the SRAM **10** to “become” executable.


The paragraph beginning on page 10 line 8 has been amended as follows:

According to a further preferred embodiment of the present invention, the suggested download algorithm, or set of instructions, and system architecture can be easily enhanced to have a better functionality. For example, the suggested architecture, as can be seen in figure 2, involves time slots when the memory is not available for execution with the required code. In these cases, while the memory device is in the process of downloading the required code for execution, it is required

to supply a "busy signal" 26 to the executing entity 3012 in order to notify that the required code is not yet available. The executing entity 3012 should use the "busy signal" 26 in order to hold off the execution attempt until the memory device is ready and able to supply the required code. There are many prior art platform-dependant methods of holding off an execution attempt.

In the claims:

Claims 1, 4, 6, 7 and 9 have been amended as follows:

 1. (Amended) A system for enabling code execution from non executable memory, comprising:

- (i[.]) [An] an executing entity, for executing code for a host system;
- (ii[.]) [A] a non-executable memory component, for storing system code and data; and[.]
- (iii[.]) [An] an executable memory component, for operating as a memory buffer for executing said code, such that a portion of contents of said non-executable memory component is located within said executable memory component, and said portion of contents of said non-executable memory component emulates executable functions of said executable memory component.

4. (Amended) A system for executing code using non-executable memory, comprising:

- (i[.]) [An] an executing entity, for executing code;

(ii[.]) [A] a non-executable memory component, for storing said code and data; and[.]

(iii[.]) [A] a plurality of executable memory components that [operate] alternate as [multiple] memory buffers, thereby [for] preventing memory lockage for accesses to said data during download operations of said code.

6. (Amended) A method for executing code using non-executable memory, comprising the steps of:

(i[.]) providing executable memory, for buffering at least one code request from an executing entity;

(ii[.]) providing a non-executable memory, for storing executable code;

(iii[.]) downloading at least a portion ^B of said executable code to said executable memory, for emulating executable functions of said executable memory;

(iv[.]) executing at least one said code request from said executable memory; and

(v[.]) buffering an execution of contents of said non-executable memory in said executable memory.

7. (Amended) The method of claim 6, further comprising the steps of:

[(a)vi] managing at least one set of instructions to guarantee availability of said contents in an executable buffer; and

~~([b]vii) supplying a busy signal in cases where said contents are not available, such that the [executable] executing entity delays the read cycle until said contents are available.~~

9. (Amended) The method of claim 6, such that step (iv) further includes:

[I.](a) providing a plurality of executable memory buffers for preventing said portion of said non-executable memory from being locked for accesses during [*said*] said downloading operation;

[II.](b) loading said executable code ~~to~~ ^{to} one of said plurality of executable memory buffers; and

[III.](c) maintaining at least one of additional said executable memory buffers, to be accessible to said [host system] executing entity and executable by said [*host system*] executing entity.

New claims 10-22 have been added as follows:

10. (New) A device for enabling an executing entity of a host system to execute code, comprising:

- (i) a non-executable memory component, for storing the code; and
- (ii) at least one executable memory component, each said executable memory component for presenting at least a portion of said stored code to the executing entity in a manner that enables the executing entity to execute said portion of said stored code directly from said each executable memory.

11. (New) The device of claim 10, wherein said non-executable memory component and said at least one executable memory component are separate from the host system.

12. (New) The device of claim 10, further comprising:

(iii) a mechanism for guaranteeing availability, in one of said at least one executable memory component, of code requested by the executing entity.

13. (New) The device of claim 10, comprising a plurality of said executable memory components, such that while one said executable memory component is presenting a first said at least portion of said stored code to the executing entity, a second said at least portion of said stored code is being downloaded to another said executable memory component.

14. (New) The device of claim 10, wherein each said at least one executable memory component is too small to accommodate all of the code at once.

15. (New) A method of executing code, comprising the steps of:

(a) storing the code in a non-executable memory component;

(b) downloading at least a portion of the code from said non-executable memory component to a first executable memory component; and

(c) executing said downloaded code, by an executing entity of a host system, said first executable memory component being separate from said host system.

16. (New) The method of claim 15, wherein said executing entity executes said downloaded code directly from said first executable memory component.

17. (New) The method of claim 15, wherein only a first portion of the code is downloaded to said first executable memory component.

18. (New) The method of claim 17, further comprising the steps of:

- (d) subsequent to said downloading, requesting code to be executed, by said executing entity;
- (e) if said requested code is outside of said downloaded first portion of the code:
 - (i) downloading a second portion of the code, including said requested code, from said non-executable memory component to said first executable memory component; and
 - (ii) during said downloading of said second portion of the code, suspending activity of said executing entity.

19. (New) The method of claim 18, wherein said suspending includes supplying a busy signal to said executing entity.

20. (New) The method of claim 16, further comprising the steps of:

- (d) downloading a second portion of the code to a second executable memory component; and
- (e) executing said downloaded second portion of the code, by said executing entity.

21. (New) The method of claim 20, wherein said second executable memory component is separate from said host system.

22. (New) The method of claim 21, wherein said executing entity executes said downloaded second portion of the code directly from said second executable memory component.